

Generative container implementation in C++

presented at

GCSE Young Researchers Workshop

Erfurt, September 11th, 2001

by

Andreas P. Priesnitz

Institute for
Numerical and Applied Mathematics
Georg–August–University Göttingen

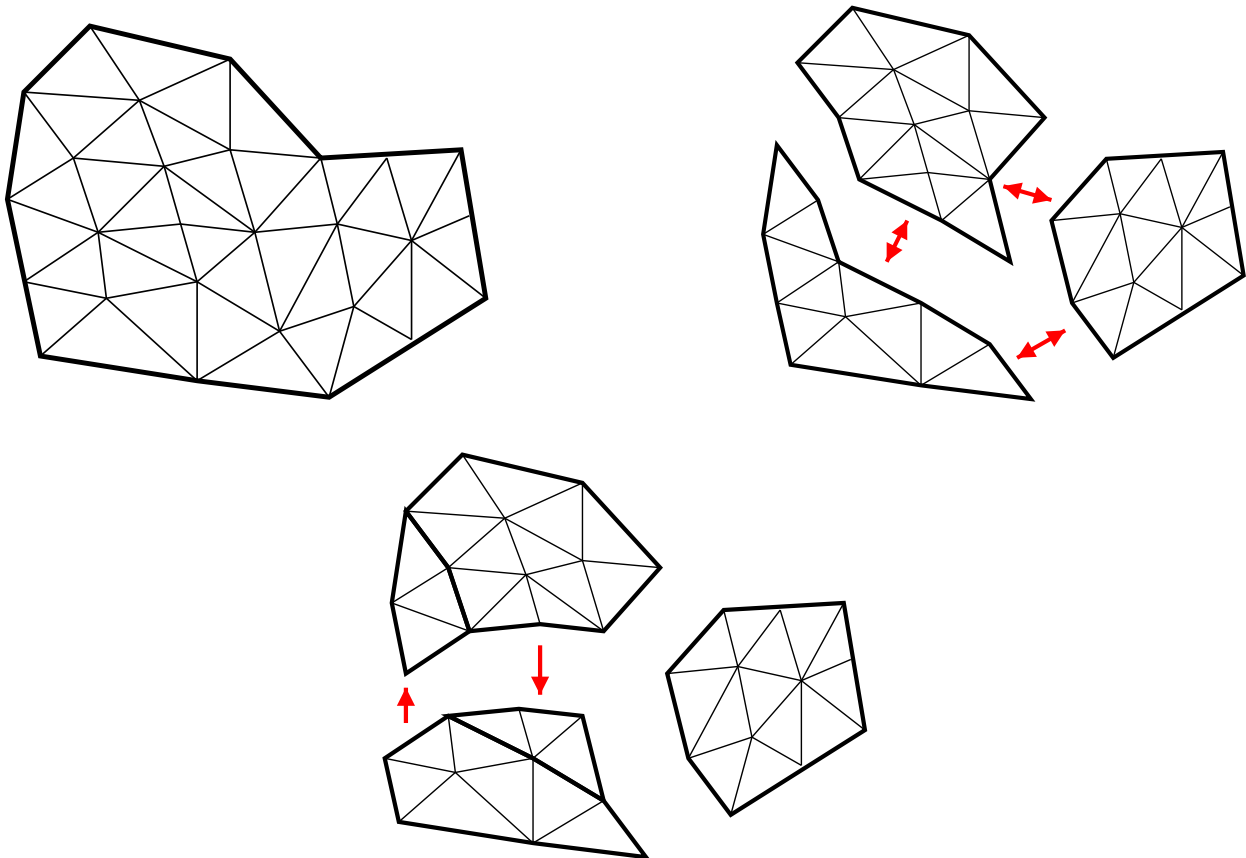
`priesnit@math.uni-goettingen.de`

`http://www.num.math.uni-goettingen.de/priesnit`

Motivation

Computational Fluid Dynamics

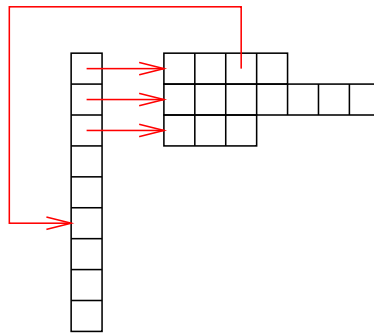
- ↪ Adaptive Finite Element Methods
- ↪ Domain Decomposition
- ↪ Load Balancing
- ↪ ...
- ↪ **Implementation of a grid data structure that efficiently supports such algorithms**



Approach

Mesh structures are interpreted by various *weighted graphs* with points or simplices as vertex weights and profoundly different features.

Graphs need to efficiently provide these features. They are implemented by appropriate *adjacency lists*.



Lists are containers, i.e. sets of stored items that can be traversed in some reproducible order.

Optionally, this order is permutable ("linked list").

Allocation may be *static* or *dynamic*, *nodewise* (not supported yet) or *blockwise* and then *constant* or *reallocable*.

STL container concept extensions

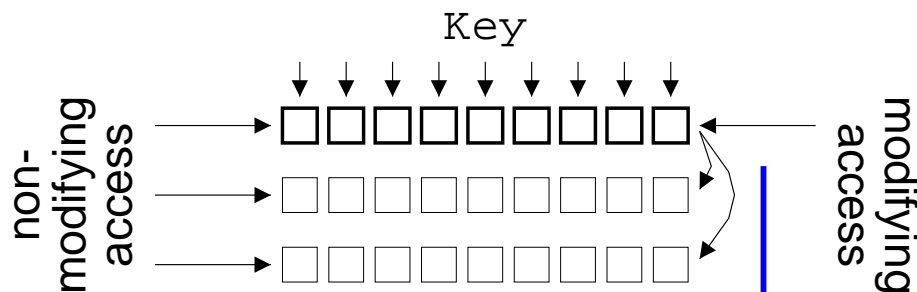
Positional information is modeled **invariantly** by an **item type independent** Key rather than iterators.

Any container may be interpreted as a functor

$$\text{Key} \longrightarrow R$$

where R is a reference-to-item type.

Optionally, *dependent* containers of identic structure, but any item type, can be generated from a *replicable* container. Changes to that are automatically applied to its dependants, which cannot be changed actively.

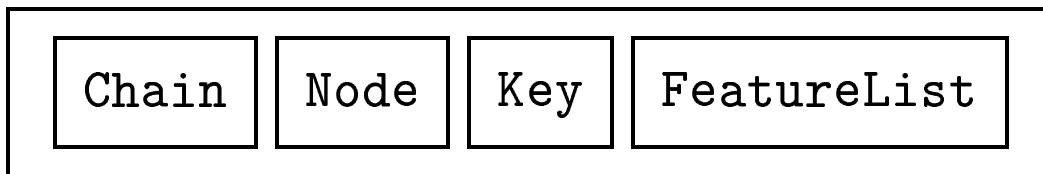


This way **dynamic** association of data to items of an existing container (e.g. temporary colouring in a graph algorithm) is efficiently facilitated. Key objects allow simultaneous access to the associated containers.

Technique

Generic programming and the *mixin layer* approach allow efficient yet separate implementation of features.

Each layer refines the involved types, implementing some feature:



`FeatureList` holds information on the implemented features as a list of types.

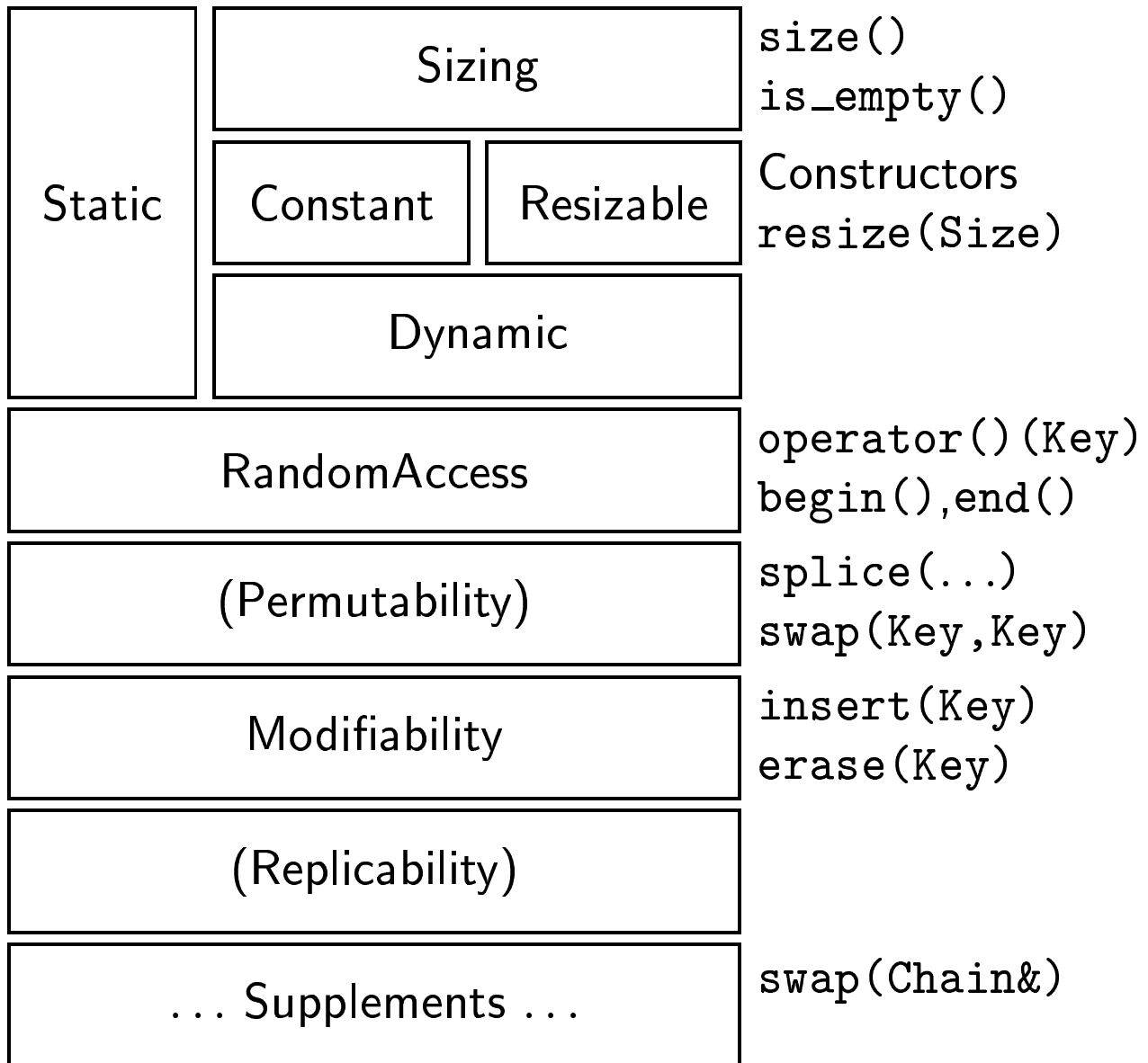
Generative programming techniques compose the layers:

Alternative layers are selected within feature implementations according to user-supplied configuration parameters.

A generator stacks the layers and defines the types in the correct order, passing each as an argument to the next.

Implementation

The layer hierarchy, in bottom–up order, is given with functions representative for the layers' features



Application

The container generator's interface is:

```
template <
    typename Element_,
    typename Modifiability_,
        // Fixed<size_t>
        // | Constant
        // | Modifiable
    typename Permutability_,
        // Linear | Permutable
    typename Replicability_,
        // Independent | Replicable
    typename Optimizing_,
        // NoOpt | Opt<KEY,ITERATOR>
    typename Checking_
        // NoCheck | Check<RANGE,...>
>
struct Generator
{
    typedef ... Type;
};
```

Outlook

Next steps will be:

- Finishing and testing the implementation
- Application of techniques to graph and mesh design
- Extension and refinement of container implementation according to practical experiences